Embedding Linkages on an Integer Lattice

Susan Landau*
MS UBUR02-313
Sun Microsystems Laboratories
P.O. Box 4002
Burlington MA 01803-0902
susan.landau@east.sun.com

Neil Immerman†
Computer Science Dept.
University of Massachusetts
Amherst, MA 01003
immerman@cs.umass.edu

Abstract

This paper answers the following question: Given an "erector set" linkage, a connected set of fixed-length links, what is the minimal ϵ needed to adjust the edge lengths so that the vertices of the linkage can be placed on integer lattice points? Each edge length is allowed to change by at most ϵ. Angles are not fixed, but collinearity must be preserved (although the introduction of new collinearities is allowed). We show that the question of determining whether a linkage can be embedded on the integer lattice is strongly $\mathcal{NP}$-complete. Indeed, we show that even with $\epsilon = 0$ (under which the problem becomes "Can this linkage be embedded?"), the problem remains strongly $\mathcal{NP}$-complete. However for some applications, it is reasonable to assume that lengths of the links and the number of "co-incident" cycles are bounded (two cycles are co-incident if they share an edge). We show that under these bounding assumptions, there is a polynomial time solution to the problem.

1 Introduction

In layout on a computer screen, location – especially location at an integer-grid point – can be more important than getting lengths exactly right. One can model such a situation by building an "erector set" type linkage that uses straight, fixed-length links and joints. Angles are free to change, but lengths

*Supported by NSF grants CCR-9204630 and CDA–9753055.
†Supported by NSF grant CCR-9505446

and collinearity properties of the linkage are fixed. A problem in computer-aided design [8] raised the question of when a linkage can be embedded on an integer grid (that is with its vertices at integer lattice points), or if the linkage could not be so embedded, what was the minimal ϵ needed to ensure that the vertices of the linkage are embedded on integer lattice points if one is allowed to shrink or stretch lengths of the links by ϵ. Graph edges are allowed to cross and even to coincide, but the underlying connectivity properties of the graph must be unchanged under this ϵ transformation.

Suppose edges are allowed to stretch or shrink by a given ϵ. We show that the question of determining whether a linkage can be embedded on the integer lattice is $\mathcal{NP}$-complete. Indeed, we show that even with $\epsilon = 0$ (under which the problem becomes "Can this linkage be embedded?"), the problem remains $\mathcal{NP}$-complete. However for some applications, it is reasonable to assume that lengths of the links and the number of "co-incident" cycles are bounded (two cycles are co-incident if they share an edge). We show that under these bounding assumptions, there is a polynomial time solution to the problem.

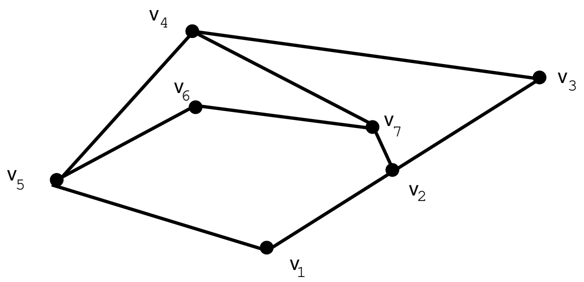


Figure 1: A Linkage with Eight Links and Seven Vertices.

This problem has obvious applications to displaying and printing of diagrams. Similarly, in fabrication of many kinds of materials including circuits or machinary, it is convenient to have certain points of the design placed on evenly spaced grid points. Similar issues come up with robot placement and assembly by robot.

We note that the problem has some obvious ties to number theory: a linkage can be embedded only if each of its links can be embedded, and the latter can happen if and only if each of the links has a length whose square can be written as the sum of two squares. We explore these connections in the paper.

This paper is organized as follows: §2 Embedding Fixed Length Linkages, §3 Embedding Linkages with Restrictions, and §4 What is the minimal ϵ?.

2 Embedding Fixed Length Linkages

A linkage $\mathcal{L}$ is defined as a connected graph with fixed-length edges. A link is an edge of finite length. By abuse of notation, the lengths of each of the links will be denoted by the same symbol that is used to denote the edge: $l_1, \ldots, l_n$. Links must be of positive length, and each link l_i has two distinct endpoints, c_i and d_i. A link may also contain labelled intermediate points. In that case, the vertex which lies within another link is collinear with the endpoints of the link in which it lies. For example, in Figure 1, the link (v_1, v_2, v_3) must consist of three collinear points, and the distances (v_1, v_2) and (v_2, v_3) are fixed.

An arbitrary number of links may connect to a single vertex. We say a linkage can be embedded on an integer lattice if there is a physical realization of the linkage that preserves collinearity and connectedness properties while embedding the linkage on the lattice in such a way that the vertices are placed on integer lattice points. (It is no problem if the link also passes through a lattice point.) We say a linkage can be ϵ-embedded on an integer lattice if there is another linkage $\mathcal{L}'$ such that for each link l_i in $\mathcal{L}$ there is a link l'_i in $\mathcal{L}'$ where $|l_i - l'_i| \leq \epsilon$, the adjacency matrices of $\mathcal{L}$ and $\mathcal{L}'$ are the same, any necessarily collinear points of $\mathcal{L}$ are collinear in $\mathcal{L}'$ and the vertices of $\mathcal{L}'$ lie on integer lattice points.

We are interested in the problem of given a linkage $\mathcal{L}$, find the minimum ϵ such that $\mathcal{L}$ can be $\epsilon-$embedded on an integer lattice in two dimensions. We will allow links to lie upon one another and to cross. In the usual way, the size of the problem is the size of the input, $\Sigma \log l_i$. We make the following observation:

Theorem 2.1 *A linkage $\mathcal{L}$ with links of length $l_1, l_2, \ldots, l_n$ can be ϵ-embedded on a two dimensional integer grid only if for each $i = 1, \ldots, n$, there is an integer n_i such that $\sqrt{n_i} \in [l_i - \epsilon, l_i + \epsilon]$ and n_i is expressible as the sum of two squares.*

Proof Obvious. □

In the remainder of this section we limit ourselves to a simpler problem: we consider the problem of embedding with $\epsilon = 0$. We begin with the following well-known characterization of which integers can be written as the sum of two squares.

Theorem 2.2 *([5], pg. 299) A number n is the sum of two squares if and only if all prime factors of n of the form $4m+3$ have even exponents in the prime factorization of n.*

We will let the symbol p_i denote a prime congruent to 1 mod 4, and q_j denote a prime congruent to 3 mod 4. Let $r_2(n)$ be the number of distinct representations of n as the sum of two squares; we count two representations (a,b) and (c,d) as distinct iff $(a,b) \neq (c,d)$. Then we have:

Theorem 2.3 *([5], pg. 242) Suppose $n = 2^{\alpha}\Pi p_i^{r_i}\Pi q_j^{s_j}$. Then $r_2(n) = 4\Pi(r_i+1)$ if each of the s_j is even, and 0 otherwise.*

The following is well-known information about the size of $r_2(n)$:

Theorem 2.4 *([5], pg. 270) $r_2(n) = O((\log n)^{\Delta})$ is false for every constant Δ.*

Theorem 2.5 *([5], pg. 270) The maximum order of magnitude for $r_2(n) = O(2^{\frac{\log n}{2\log\log n}})$.*

For simplification, we will use the approximation that $r_2(n) < n^{\Delta}$ for every constant $\Delta > 0$.

The possible number of embeddings implied by Theorem 2.4 might lead one to suspect that from a computational viewpoint, the problem of embedding a linkage in an integer lattice is hard, and indeed, it is. We study a simple linkage first.

The simplest kind of linkage is one whose underlying graph structure is a tree; by abuse of notation, we call such a linkage a tree. Then we have:

Lemma 2.6 *A linkage $\mathcal{T}$ that is a tree can be embedded if and only if each of its links can be embedded.*

An embedding is a function from the vertices to lattice points. Consider two embeddings of the linkages that map the root vertex to the origin; we say these are distinct if they are not equal as maps. Then we have:

Theorem 2.7 *Suppose the tree $\mathcal{T}$ with edges $t_1, \ldots, t_n$ can be embedded on the integer lattice, and suppose $t_i^2 = 2^{\alpha_i} \Pi_j p_{ij}^{r_{ij}} \Pi_k q_{ik}^{s_{ik}}$ with s_{ik} even for all ik. The number of distinct embeddings of $\mathcal{T}$ that map the root to the origin equals $4^n \Pi_{i,j}(r_{ij}+1)$.*

Proof We begin by embedding the root t_1 at the origin. Having decided the location of c_1, there are $4\Pi(r_{1j}+1)$ places to locate d_1. The embedding of each d_i is dependent only on the location of c_i and the length of the link. Thus the number of possible embeddings of t_i is $4\Pi(r_{ij}+1)$. The result follows immediately. $\square$

This result means that checking whether a tree can be embedded can be done in time $O(\Sigma R(t_i))$, where $R(t_i)$ is the time needed to compute whether t_i can be written as the sum of squares. At present, the fastest algorithm to check whether an integer can be written as a sum of squares is exponential time.

Indeed that problem is likely to be computationally hard. In the appendix, we show that if a positive integer n has a bounded number of prime factors of the form $4k+1$ and no prime factors of the form $4k+3$, the problem of enumerating the representations of n as the sum of two squares is polynomial-time equivalent to factoring n. There are no known subexponential algorithms for factoring general integers n. But embedding a linkage is computationally expensive not only for number-theoretic reasons.

Our proof that the problem is $\mathcal{NP}-$complete shows that the problem is strongly $\mathcal{NP}-$complete. This leads us to suspect that the embedding problem is hard computationally because of the complexity of the interaction of cycles of the linkage.

We show $\mathcal{NP}-$completeness by doing a reduction from $3-\mathcal{SAT}$. Given a formula φ in 3-$\mathcal{CNF}$, we build a linkage that can be embedded with its vertices on integer lattice points if and only if φ is satisfiable. The intuition underlying the following three lemmas is that we are building a linkage gadget that will check the truth value of a clause; we will then join the gadgets together to model the $3-\mathcal{CNF}$ formula. We conclude this section by showing that the problem is $\mathcal{NP}-$complete.

Let φ be a $3-\mathcal{CNF}$ formula $\varphi = C_1 \wedge \ldots \wedge C_r$ where each $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$, and the l_{ij}'s are literals.

We begin by building a gadget G_i for each clause $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$. The gadget G_i consists of two parts. The first part, drawn on the left in Figure 2, consists of three connected diamonds. The line from y_{i0} to y_{i6}

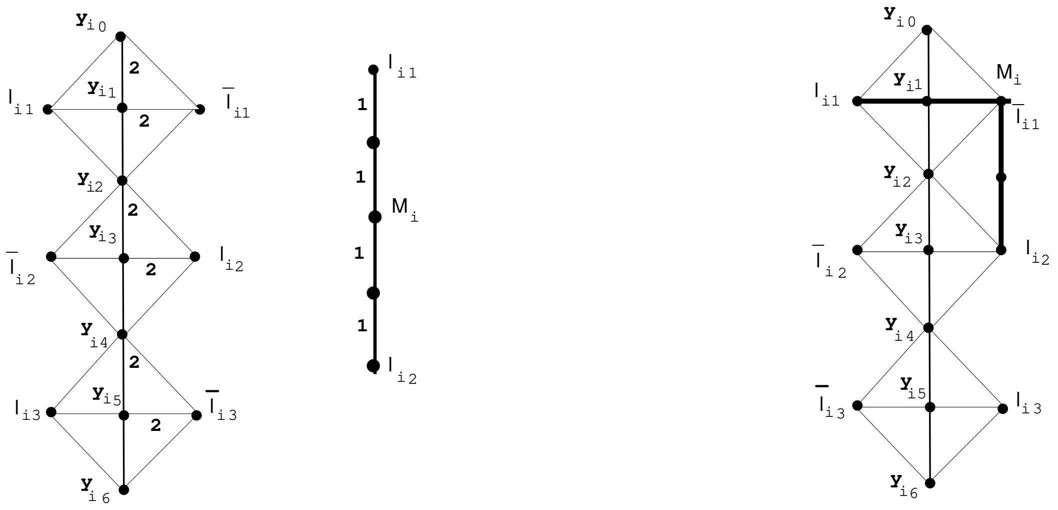


Figure 2: Gadget G_i, on the right, consists of the combination of the two gadgets on the left, where points with the same label are co-incident.

is a straight line of length 6. The second part of the gadget is a linkage connecting l_{i1} with l_{i2}. The length of the diagonals of the diamonds is 2. The linkage connecting l_{i1} and l_{i2} consists of four line segments of unit length. Its midpoint is labelled M_i. Without loss of generality, the linkage can be arranged so that the vertices y_{ij} are on the $y-$axis. A useful intuition is that placing l_{ij} on the line $x = 1$ is assigning l_{ij} the value "true," while placing l_{ij} on the line $x = -1$ is assigning l_{ij} the value "false."

This link can be folded in various ways. The gadget is constructed so that if G_i is placed with vertices y_{ij} on the $y-$axis, then the vertex M_i can be on the line $x = 1$ if and only if l_{i1} or l_{i2} lies on the line $x = 1$.

Thus G_i is almost an or gate in that its "output" M_i can lie on the line $x = 1$ iff at least one of l_{i1} or l_{i2} lies on the line $x = 1$. It is convenient to force the output of this "or" gate to have a fixed $y-$coordinate. We do this by extending G_i to H_i whose output P_i has the same x coordinate as M_i but whose $y-$coordinate is fixed.

The gadget H_i is a combination of G_i with a "ladder." The ladder connects the vertices y_{ij} with the vertex M_i and a new vertex P_i. The sides of the ladder are 1, the diagonals are length $\sqrt{2}$. If H_i is placed with the y_{ij} vertices on the $y-$axis, then M_i and P_i are connected in such a way as to ensure that they have a common $x-$value. Thus P_i is on the line $x = 1$

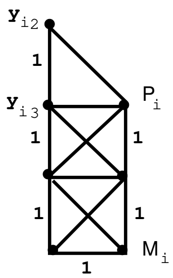
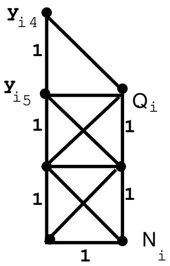


Figure 3: Gadget H_i consists of the above combined with Gadget G_i, where points with the same label coincide.



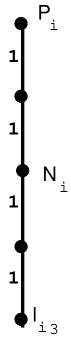


Figure 4: Gadget K_i, consisting of the above combined with Gadget H_i, where points with the same label coincide.

if and only if l_{i1} or l_{i2} have their $x-$value equal to 1.

H_i is a true or gate. We combine this with a similar construction to build a three-way or-gate. For completeness we show this entire three-way or-gate, K_i, in Figure 4. Note that its output Q_i has its $y-$coordinate fixed level with y_{i5} and that Q_i can be placed on the line $x = 1$ iff at least one of l_{i1}, l_{i2} or l_{i3} lies on the line x=1.

The gadget K_i, is a combination of H_i with another ladder and another straight piece (similar to the one used in G_i). The straight piece adds another special vertex N_i, which connects P_i and l_{i3}. Again we are assuming that the gadget is arranged so that the y_{ij} are on the $y-$axis. Then N_i can be placed on the line $x = 1$ if and only if P_i or l_{i3} are. Since P_i is on the line $x = 1$ if and only if l_{i1} or l_{i2} are, N_i has $x-$value equal to 1 if C_i evaluates to "true." The ladder in K_i ensures that Q_i has the same $x-$value as N_i.

The completed gadget K_i works as follows: without loss of generality, we can assume that K_i is aligned on the integer lattice so that the y_{ij} vertices lie on the $y-$axis. Then Q_i lies on the line $x = 1$ if and only if l_{i1} has $x = 1$ or l_{i2} has $x = 1$ or l_{i3} has $x = 1$.

We now are ready to build the linkage $\mathcal{L}(\varphi)$. Each clause C_i of φ has a corresponding gadget K_i, built as described above. We line up all the gadgets K_i one underneath the other, the top of K_{i+1} connecting with the bottom of K_i. We need to ensure that all appearances of a literals have the same truth value. This is done by connecting appearances of the same variable with a link as follows: if the literal l_{is} is the same as the literal l_{jt}, and either the literal l_{is} or the literal l_{jt} is not already connected to other literals, then we connect the two instances, l_{is}, l_{jt}, with a link of length of $|6(j-i)+4(t-s)|$. This length ensures the literals l_{is} and l_{jt} will have the same $x-$value. (Note that if we want to keep all our lines short, we may replace this line by a system of $|6(j-i)+4(t-s)|$ connected, unit-length links.)

Next we add a linkage consisting of r links of length 6, connecting vertices $Q_1, \ldots, Q_r$. This assures that all the clauses are true simultaneously.

It is clear that a satisfying assignment of the variables leads to an embedding of the linkage. It is also clear that *any* embedding of the linkage can be transformed to one that places the y_{ij} on the $y-$axis *and* the Q_i on the line $x = -1$ *or* $x = 1$. If the Q_i are all on the line $x = 1$, then the l_{ij} on the line $x = 1$ are assigned to "true," and we have a satisfying assignment. If the Q_i are all on the line $x = -1$, assign the literals on the line $x = -1$ the value "true," and again we have a satisfying assignment. Furthermore, the reduction we have presented is in polynomial time. More formally,

Lemma 2.8 *For each $C_i = l_{i1} \vee l_{i2} \vee l_{i3}$, where the l_{ij}'s are literals, the gadget G_i with the vertices y_{ij} on the y-axis can have the vertex M_i on the line $x = 1$ if and only if l_{i1} or l_{i2} is on the line $x = 1$.*

Lemma 2.9 *With conditions as above, the gadget H_i can be embedded on the integer lattice. P_i can be on the line $x = 1$ if and only if l_{i1} or l_{i2} is on the line $x = 1$.*

Proof The first statement is clear. The second statement follows from the fact that of necessity P_i and M_i have the same x value. □

Lemma 2.10 *The gadget K_i can be embedded on the integer lattice. The point Q_i can be on the line $x = 1$ if and only if C_i evaluates to true (where the literals in C_i are assigned true iff they lie on the line $x = 1$).*

Let Linkages be the problem of whether a linkage can be embedded on the integer lattice. We have shown:

Theorem 2.11 *Linkages is strongly $\mathcal{NP}$-complete.*

It is interesting to compare this result with an earlier theorem of Hopcroft, Joseph and Whitesides [7]. Their result shows the ruler problem — the question of whether a carpenter's ruler consisting of a sequence of n links of integer lengths, $l_1, \ldots, l_n$, hinged together at the endpoints, and allowed to fold so that angles of either $0°$ or $180°$ are formed, can be aligned so that its length is k — is $\mathcal{NP}-$complete. The proof is via a reduction from the Partition problem. The reduction does not show the ruler problem to be strongly $\mathcal{NP}-$complete. In fact, it is not. In [7], there is a dynamic programming solution to the problem that gives a polynomial-time solution if the lengths in the ruler problem are polynomially-bounded.

The reduction we have used to prove the strong $\mathcal{NP}-$completeness of Linkages shows that the problem remains $\mathcal{NP}$-complete even when the lengths of all the links are bounded by a fixed constant. The complexity arises instead from the interactions of the polygons. There are two useful restrictions that simplify the problem computationally but still include the class of problems we are interested in in practice. In the next section we consider these.

3 Embedding Linkages with Restrictions

The first restriction is to require that the linkage be of bounded size, that is, the linkage must lie within an $m \times m$ integer grid. (Obviously this implies that all of the lengths l_i must be small.) The second restriction is combinatorial: it is that there are only a bounded number of closed figures – "polygons" – in the linkage. For many applications that is a very reasonable assumption. With these restrictions, determining whether the linkage can be embedded with its vertices on integer lattice points can be done in polynomial time.

We define a subgraph, $\mathcal{P} = l_1, \ldots l_n$, of a linkage to be a polygon if the set of vertices in $\mathcal{P}$ plus its induced set of edges form a cycle with $c_1 = d_n$. There are potentially exponentially many embeddings, but the yes-no question of whether a polygon can be embedded on an $m \times m$ grid can be answered in polynomial time.

Theorem 3.1 *For any $\Delta > 0$, we can check if the polygon $l_1, \ldots, l_n$ can be embedded in an $m \times m$ grid in time $O(nm^{4+\Delta})$. We can compute whether this polygon can be embedded starting at a given initial point (k, l) in time $O(nm^{2+\Delta})$.*

Proof Let λ denote the empty string. In the algorithm below we compute the following set of matrices, for $i = 0, \ldots, n$ for all initial points (k, l):

$$M^{k,l}_{l_1 \ldots l_i}(s,t) = \begin{cases} 1 & \text{if linkage } l_1 \ldots l_i \text{ can be embedded} \\ & \quad \text{starting at} (k,l) \text{ and ending at} (s,t) \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 3.1:
Input: polygon $\mathcal{P} = l_1, \ldots, l_n$, integer m
Output: 1 iff $\mathcal{P}$ can be embedded in the $m \times m$ grid.

1. ans := 0
2. for all $(k, l, s, t) \in \{1, \ldots, m\}$ do in parallel {
3. $\quad M^{k,l}_{\lambda}(s,t) = \begin{cases} 1 & \text{if } (k,l) = (s,t) \\ 0 & \text{otherwise} \end{cases}$
4. $\quad$ for $i = 1, \ldots, n$ do:
5. $\quad\quad M^{k,l}_{l_1 \ldots l_i}(s,t) = \begin{cases} 1 & \text{if } \exists (u,v) \text{ reachable from } (s,t) \text{ via } l_i \\ & \quad \text{such that } M^{k,l}_{l_1 \ldots l_{i-1}}(u,v) = 1 \\ 0 & \text{otherwise} \end{cases}$
6. $\quad$ if $M^{k,l}_{l_1 \ldots l_n}(k,l) = 1$ then ans := 1 }
7. return(ans)

It is easy to show by induction on i that Algorithm 3.1 correctly computes the matrices $M^{k,l}_{l_1 \ldots l_i}$. Thus the algorithm produces the correct answer.

The running time is nm^4 times the time to compute step 5 for a fixed k, l, s, t, and i. Point (s, t) is reachable from at most $r_2(l_i^2)$ points (u, v) via link l_i. Thus the time for this step is bounded by $O(r_2(l_i^2))$.

We know from Theorem 2.5 that for sufficiently large l_i, $r_2(l_i^2) < (l_i^2)^{\Delta/3}$. Also note that $l_i^2 < (\sqrt{2}m)^2 = 2m^2$. Thus for sufficiently large m,

$$r_2(l_i^2) \; < \; \left((\sqrt{2}m)^2\right)^{\Delta/3} \; < \; m^\Delta$$

Once and for all for each l_i, we need to factor l_i and compute all the $r_2(l_i^2)$ possible placements of l_i starting at the origin. This adds a cost of less than $O(nm)$ to the total running time. Thus the total running time is $O(nm^{4+\Delta})$ as desired. $\square$

Note: There are symmetries that one can use to speed up this computation (e.g., one need only try one-quarter of the grid for a possible starting point). Furthermore, the linkage can be embedded in an $m \times m$ grid only if the linkage can be embedded in a $2m \times 2m$ grid starting at the point (m, m). One can check whether the polygon is embeddable in a $2m \times 2m$ grid in $O(nm^{2+\Delta})$ steps, by using the point (m, m) as a starting point.

In Algorithm 3.1 we determine whether the polygon $\mathcal{P} = l_1, \ldots, l_n$ can be embedded by starting at some point (k, l) and seeing if we can embed the whole polygon, ending again at (k, l). It is convenient to then back around the whole polygon computing the lattice points at which each vertex of $\mathcal{P}$ may be embedded.

To determine where the vertex c_i beginning link l_i can be embedded we simply take the conjunction of the matrices $M^{k,l}_{l_1,\ldots,l_{i-1}}$ and $M^{k,l}_{l_n^r,l_{n-1}^r,\ldots,l_i^r}$, where l_j^r means the reversal of link l_j. Define the matrix $G^{kl}_{c_i}$ to be this conjunction,

$$G^{kl}_{c_i}(s,t) \quad = \quad M^{k,l}_{l_1,\ldots,l_{i-1}}(s,t) \text{ and } M^{k,l}_{l_n^r,l_{n-1}^r,\ldots,l_i^r}(s,t) \tag{3.2}$$

It follows from the proof of Theorem 3.1 that

Corollary 3.3 *Given polygon $\mathcal{P} = l_1, \ldots, l_n$, integer m, and lattice point (k, l) we can compute all the boolean matrices $G^{kl}_{c_i}$ in total time $O(nm^{2+\Delta})$. $G^{kl}_{c_i}(s,t) = 1$ iff c_i can be placed on lattice point (s, t) when cP is embedded in the $m \times m$ lattice with c_1 placed at (k, l).*

We also observe that the bounds of Theorem 3.1 and Corollary 3.3 remain true when we are given additional information restricting the location of certain vertices.

Corollary 3.4 *Suppose that we are given a polygon as in Theorem 3.1 but suppose now that we are also given additional boolean matrices A_v for some of the vertices in the polygon. Here $A_v(x, y) = 0$ means that vertex v may not be placed on lattice point (x, y). Then the computations of Theorem 3.1 and Corollary 3.3 can be completed in the same asymptotic running time.*

Proof Everything is as before, but whenever we compute $M^{k,l}_{l_1 \ldots l_i}(s, t)$ we must conjoin this with $A_{d_i}(s, t)$. Recall that d_i is the end of link l_i. □

We now consider the more complex problem of embedding a linkage that has a bounded number of polygons. It is useful to view the linkage as a graph, although normally graphs have straight edges between a pair of vertices, and in our case, the linkage may have an edge with an internal vertex, as the following example demonstrates:

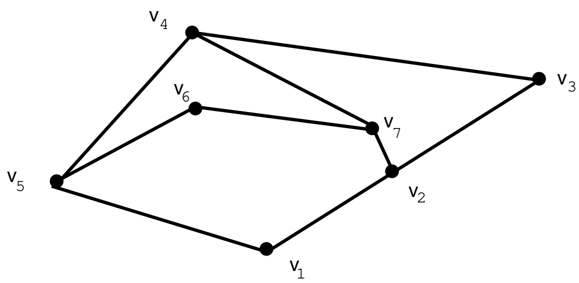


Figure 5: A Linkage with Eight Links and Seven Vertices.

The edge v_1, v_3 includes the vertex v_2; all three vertices must lie on a straight line. This will not present us with a serious difficulty, but we will postpone discussion of this issue until after we have shown how to handle embedding a linkage in an $m \times m$ lattice.

As Theorem 2.11 demonstrated, a linkage is computationally complex precisely because several polygons may share a vertex, or set of vertices; the question is how to place them on the grid so that all the polygons can be simultaneously embedded.

We now show how to embed biconnected components on an $m \times m$ grid. We assume the reader is familiar with the depth-first search algorithm for biconnected components [1]:

Theorem 3.5 *Let $\mathcal{B} = l_1, \ldots, l_b$ be a linkage which, viewed as a graph, is biconnected. Suppose we are given a depth-first search tree of $\mathcal{B}$ that has e*

back edges. Determining whether $\mathcal{B}$ can be embedded in an $m \times m$ integer grid can be done in $O(bm^{2+2e+\Delta})$ steps for any $\Delta > 0$. If we must start the embedding at a specific lattice point then the running time is $O(bm^{2e+\Delta})$

Proof Start by building the depth-first search tree for $\mathcal{B}$. Let c_1 be the end point of the first backedge that we encounter. Thus, the path from c_1 back to itself is the first polygon that we have encountered, call it $\mathcal{P}_1$. Let (k, l) be a candidate placement for vertex c_1. By Corollary 3.3 we can compute the matrices $G_{c_i}^{k,l}$ for all vertices c_i from $\mathcal{P}_1$. These tell us the possible placements of $\mathcal{P}_1$ when it is started at (k, l).

Now the situation grows more complicated. We need to check if the polygons for which we have computed embeddings can be compatibly embedded with the other polygons of the biconnected component. We can compute the matrices G_i for each of the other polygons; the question is how to combine them to determine if there is a simultaneously compatible embedding for the linkage as a whole. We make several observations:

- Each new polygon can be uniquely identified by its back edge in the depth first traversal.
- This back edge closes the section of the tree into a polygon.
- There are at most m^2 lattice points on which the endpoint of the back edge may to lie.

To check if a new polygon, $\mathcal{P}_j$, has a compatible embedding with the linkage, we consider all possible positions of the end point of its backedge c_j, and check whether any lead to an embedding. (In practice, we simply record the current position of c_j and backtrack to check that there is a consistent positioning of the endpoints of all lower-numbered backedges. The vast majority of the $m^{2(e-1)}$ conceivable placements of all these endpoints will be eliminated without backtracking very deeply. Thus the $m^{2(e-1)}$ factor in the worst case upper bound is a significant over-estimate of the true running time.)

Thus what we do is travel the dfs tree, computing all possible embeddings of each of the associated back-edge endpoints. There are e of them. Hence we have a running time of $O(bm^{2+2e+\Delta})$ steps. □

We now finish the algorithm to decide whether a general linkage, $\mathcal{L}$, is embeddable. We traverse a depth-first search tree of $\mathcal{L}$ in postorder fashion, computing the possible embeddings of each biconnected component as we go. The general linkage algorithm follows from Theorem 3.5.

Theorem 3.6 *Suppose a linkage $\mathcal{L}$ has n links and each biconnected component has at most e back edges in the depth-first search tree of $\mathcal{L}$. The problem of embedding $\mathcal{L}$ on an $m \times m$ integer lattice is computable in $O(nm^{2e+2+\Delta})$ steps, for any $\Delta > 0$.*

Proof This is now straightforward using Theorem 3.5.

First, compute the depth-first search tree of $\mathcal{L}$ including its biconnected components. Second, we traverse the tree from leaves to root, computing the matrix

$$A_v(i,j) = \begin{cases} 1 & \text{if the tree rooted at } v \text{ may be embedded at position } (i,j) \\ 0 & \textit{otherwise} \end{cases}$$

When the matrices A_v for all the vertices v below a biconnected component C have been computed, we compute the possible positions for the root of C using Theorem 3.5. In this way, we proceed up the tree. The required time is $O(bm^{2e+2+\Delta})$ for each biconnected component of size b. Thus the total is $O(nm^{2e+2+\Delta})$ as claimed. □

Up to now we have ignored the issue of links that contain vertices in their interior. A slight modification of the above algorithms handles this problem. Let l_i be a link with endpoints c and d and internal points $a_1, \ldots, a_k$. When we travel from c to a_1 along the depth-first search tree, we would normally establish the array G_{a_1} indicating possible placements of the vertex a_1 with respect to G_c. Instead we want to establish $G_{a_1}, \ldots, G_{a_k}$, G_d, indicating possible placements of the entire link. This amounts to finding possible points for d and also checking that the a_i's land on allowable lattice points. This modification does not affect the running time of the algorithm.

These results can be extended to higher dimensions, noting that:

Theorem 3.7 *([5], pp. 310-311) An integer n can be written as the sum of three squares if and only if $n \neq 4^\alpha(8m+7)$.*

Theorem 3.8 *([5], pg. 369.) Every integer can be written as the sum of four squares.*

4 What is the Minimal ϵ?

We begin with trees in two dimensions. By Theorem 2.1 and Lemma 2.6, a tree T is embeddable whenever each of its links is the square root of an integer that can be written as the sum of two squares. Clearly each link is no more than $1/2$ away from an integer, i.e., the square root of a perfect square. Can it be as much as $1/2$ away from the square root of an integer that can be written as sum of two squares? Can it be the case that there is an integer m such that there are no integers between m^2 and $(m+1)^2$ that can be written as the sum of two squares?

Theorem 4.1 *(Bambah and Chowla, [2]) Let s_i be the i^{th} integer which can be expressed as the sum of two squares. Then $s_{n+1} - s_n = O(s_n^{1/4})$.*

This theorem will answer the case for embedding trees.

Theorem 4.2 *Let $\mathcal{T} = t_1 \ldots t_n$ be a tree. Then the minimal ϵ such that $\mathcal{T}$ can be ϵ-embedded in the integer lattice can be determined in $O((t_1)^{3/4} + \ldots + (t_n)^{3/4})$ steps.*

Proof By Theorem 2.1, it suffices to consider each link separately. Suppose link t cannot be embedded. This means t^2 cannot be expressed as the sum of two squares.

Let t' be the integer closest to t^2; we have $|t' - t^2| \leq 1/2$. By Theorem 4.1, there is an integer r such that $|r - t'| < O((t')^{1/4})$ such that r can be expressed as the sum of two squares. It takes time $O(t^{1/2})$ to deterministically test if t' can be embedded (that is the time it takes to factor (t')). Thus it takes $O(t^{3/4})$ time to find the closest integer to t which can be expressed as the sum of two squares. The minimal ϵ is the maximum of these closest distances. The time to compute the minimal ϵ for the entire tree is $O((t_1)^{3/4} + \ldots + (t_n)^{3/4})$. □

The problem of embedding a linkage is $\mathcal{NP}$-complete, thus the best we can realistically hope for is an exponential-time algorithm for the problem. The obvious algorithm of trying all possibilities is single exponential, so we have no more to say about the problem of embedding arbitrary linkages. However, there is a nice solution to linkages with a bounded number of cycles limited to an $m \times m$ grid. There are two cases to consider, one where an upper bound ϵ is given, the other not. They are handled similarly, with the latter treated first.

We begin with a theorem of Edmund Landau, then follow with a generalization of Theorem 3.1.

Let $N_2(n)$ be the number of ways to represent the integers $x \leq n$ as the sum of two squares.

Theorem 4.3 *(E. Landau, [4], pg. 22)* $N_2(x) = b\frac{x}{\sqrt{\log x}} + o(\frac{x}{\sqrt{\log x}})$, *where* $b = \frac{1}{2}\Pi_{q\equiv 3 \pmod 4}(1-q^{-2})^{1/2} \sim 0.764\ldots$.

Theorem 4.4 *Suppose a linkage $\mathcal{L}$ is a cycle $\mathcal{P}$=$l_1 \ldots l_n$. Let L =$max_i l_i^2$. Determining the minimal ϵ for which $\mathcal{L}$ can be ϵ-embedded can be determined in $O(nm^5 \log m)$ steps.*

Proof We will do the same algorithm as we did for Theorem 3.1, but we will do it many more times.

Without loss of generality, we can assume we are interested in computing an $\epsilon \leq cm$ for some constant $0 < c \leq \sqrt{2}$.

We do binary search to find the minimal ϵ such that $\mathcal{L}$ can be ϵ-embedded. Clearly $\mathcal{L}$ can be embedded with $\epsilon = \sqrt{2}m$. We check if $\mathcal{L}$ can be embedded with $\epsilon = \frac{\sqrt{2}m}{2}$. In this way, in $O(\log m)$ rounds, we will find the minimal ϵ by which the polynomial can be embedded, if there is one less than $\sqrt{2}m$.

Suppose we are checking whether the polygon can be ϵ_i-embedded. We create the array M^{kl} just as we did in the proof of Theorem 3.1, except that we mark *all* the grid points reached by a linkage of length in the interval $[l_i - \epsilon_i, l_i + \epsilon_i]$. How many grid points are marked?

By Theorem 4.3, $N_2(l_i + \epsilon_i) - N_2(l_i - \epsilon_i) = O(\frac{l_i+\epsilon_i}{\sqrt{\log(l_i+\epsilon_i)}} - \frac{l_i-\epsilon_i}{\sqrt{\log(l_i-\epsilon_i)}}) < O(\frac{2\epsilon_i}{\sqrt{\log(l_i-\epsilon_i)}}) < O(m)$ since $\epsilon_i \leq \sqrt{2}m$.

How long does this procedure take? At each step of the algorithm, the marking algorithm now has the upper bound of $O(m^3)$ per array. We may have to run the entire algorithm as much as $O(\log m)$ times for each of the m^2 starting points. Thus we get the running time of $O(nm^5 \log m)$ steps.

□

Theorem 3.5 also generalizes easily.

Theorem 4.5 *Suppose a linkage $\mathcal{L}$= $l_1 \ldots l_n$. Suppose $\mathcal{L}$ has at most e back edges per biconnected component of the dfs tree. Determining the minimal ϵ for which $\mathcal{L}$ can be ϵ-embedded in an $m \times m$ integer grid can be done in $O(nm^{2e+3} \log m)$ steps.*

Proof The proof is a natural combination of the ideas of Theorems 3.5 and 4.4. □

We observe that since $s_{n+1} - s_n = O(s_n^{1/4})$, the case with relative error can be handled in a similar way.

Acknowledgements This problem was posed by Maharaj Mukherjee at the Stony Brook Computational Geometry Workshop. Thanks to Hendrik W. Lenstra, Jr. for pointing out an error in an earlier version of this paper.

References

[1] A. Aho, J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.

[2] R.P. Bambah and S. Chowla, On Numbers which can be Expressed as the Sum of Two Squares, *Proc. Nat. Inst. Sci. India (1947)*, pp. 101-103.

[3] W. Bosma and M.P.M. van der Hulst, *Primality proving with cyclotomy*, Proefschrift, Universiteit van Amsterdam, Amsterdam 1990.

[4] E. Grosswald, *Representations of Integers as the Sums of Squares*, Springer-Verlag, 1985.

[5] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, Oxford University Press, 1971.

[6] J. Hopcroft, D. Joseph, and S. Whitesides, "Movement Problems for 2-Dimensional Linkages", pp. 282-303, in *Planning, Geometry and Complexity*," by J. Schwartz, M. Sharir and J. Hopcroft, Ablex, 1987.

[7] J. Hopcroft, D. Joseph, and S. Whitesides, "On the Movement of Robot Arms in 2-Dimensional Bounded Regions," pp. 304-329, in *Planning, Geometry and Complexity*," by J. Schwartz, M. Sharir and J. Hopcroft, Ablex, 1987.

[8] M. Mukherjee and G. Nagy, Collinearity Constraints on Spatial Subdivision Algorithms with Finite Precision Arithmetic, *Proceedings Fifth International Symposium on Spatial Data Handling*, pp. 425-431.

[9] R. Schoof, Elliptic Curves over Finite Fields and Computation of Square Roots Mod P, *Math. Comp. 44 (1985)*, pp. 483-494.

A Appendix

In this section, we show that:

Theorem A.1 *Let n be a positive integer with a bounded number of prime factors of the form $4k+1$, and no prime factors of the form $4k+3$. Then the following problems are polynomial-time equivalent:*

1. *Enumerating the representations of n as the sum of two squares.*

2. *Factoring n.*

Proof We first observe that if n has a bounded number of prime factors of the form $4k+1$ and no prime factors of the form $4k+3$, then the number of representations of n as the sum of squares is polynomial in $\log n$. This is because if

$$n = 2^{\alpha}\Pi_{i=1}^{d} p_i^{r_i},$$

then

$$N_2(n) = 4\Pi_{i=1}^{d}(r_i+1) \leq 4((\max\ r_i)+1)^d.$$

But the maximum r_i is less than or equal to $\log r$, thus

$$N_2(n) \leq 4(\log n + 1)^d.$$

Since d is bounded, the number of representations of n as a sum of squares is polynomial in $\log n$.

Now we prove that (1) is polynomial time reducible to 2. Since we can factor, the only issue is how to find a representation of each $4k+1$ prime p as a sum of two squares, $p = x^2 + y^2$. This way is easy due to a result of Schoof [9], who gave a polynomial time algorithm for finding square roots mod p. Thus one can quickly find an x such that $x^2 + 1 \equiv 0 \pmod{p}$, and therefore $x^2 + 1 = mp$ for some integer m. But then $\gcd(x+i, p)$ in $Z[i]$ gives an $a+bi$, where $p = a^2 + b^2$. Hence, (1) can be reduced to (2) in polynomial time.

As for (2) is polynomial-time reducible to (1), we say two representations of n as the sum of two squares, $a^2 + b^2 = c^2 + d^2 = n$ are inequivalent, if $a \neq \pm c, \pm d, b \neq \pm d, \pm c$, with $\gcd(a,b) = \gcd(c,d) = 1$. Euler was the first to observe that two inequivalent representations of n as the sum of two squares leads to a factorization of n. A more modern treatment is in Bosma's thesis [3]. Using the language of complexity, Bosma shows that the reduction can be done in polynomial time. □